

Bloom maps

David Talbot*

John Talbot†

February 5, 2008

Abstract

We consider the problem of succinctly encoding a static map to support approximate queries. We derive upper and lower bounds on the space requirements in terms of the error rate and the entropy of the distribution of values over keys: our bounds differ by a factor $\log e$.

For the upper bound we introduce a novel data structure, the *Bloom map*, generalising the Bloom filter to this problem. The lower bound follows from an information theoretic argument.

1 Introduction

The ability to query a map to retrieve a *value* given a *key* is fundamental in computer science. As the universe from which keys are drawn grows in size, information theoretic lower bounds imply that any data structure supporting error-free queries of a map requires unbounded space per key. However, if we are willing to accept errors, constant space per key is sufficient.

For example, in information retrieval we may wish to query the frequencies (values) of word sequences (keys) in documents. *A priori* these sequences are drawn from a universe that is exponential in the length of a sequence. Returning an incorrect value for a small proportion of queries may be acceptable, if this enables us to support queries over a far larger data set.

Consider a map consisting of n key/value pairs $M = \{(x_1, v(x_1)), (x_2, v(x_2)), \dots, (x_n, v(x_n))\}$, where the keys $X = \{x_1, x_2, \dots, x_n\}$ are drawn from a large universe U and each value $v(x)$ is drawn from a fixed set of possible values $V = \{v_1, v_2, \dots, v_b\}$. Suppose further that the distribution of values over keys is given by $\vec{p} = (p_1, p_2, \dots, p_b)$. Thus if $X_i = \{x \in X \mid v(x) = v_i\}$ then $|X_i| = p_i n$.

We consider the problem of constructing a space-efficient data structure supporting queries on M . For any key $x \in U$ the data structure should return the associated value $v(x)$ if $x \in X$, otherwise (i.e. if $x \in U \setminus X$) it should return $\perp \notin V$.

Using an information theoretic argument, we derive lower bounds on the space required to solve this problem when errors are allowed. These lower bounds are in terms of the error rate and the entropy of the distribution of values over keys $H(\vec{p})$.

We introduce the *Bloom map*, a data structure generalising the Bloom filter [1] to the approximate map problem. The space requirements of this data structure are within a $\log e$ factor of the lower bound. To be precise for an error rate of ϵ the Bloom map uses $\log e(\log 1/\epsilon + H(\vec{p}))$ bits per key.

*Department of Informatics, University of Edinburgh, EH8 9LE, Scotland, UK. Email: d.r.talbot@sms.ed.ac.uk

†Department of Mathematics, University College London, WC1E 6BT, UK. Email: talbot@math.ucl.ac.uk.

To our knowledge, this paper is the first to make use of the distribution of values over keys to analyse the approximate map problem. Moreover, the Bloom map is the first data structure to take advantage of this distribution to save space by using variable length codes for distinct values. In the many practical settings where distributions with low entropy are encountered we expect the Bloom map to be of significant interest.

The main prior work on the approximate map problem is the Bloomier filter introduced by Chazelle et al. [3]. To store key/value pairs with values drawn from a range of size b with false positive probability ϵ , the Bloomier filter requires $\alpha(\log 1/\epsilon + \log b)$ bits per key effectively using a fixed width encoding for any value in the range. It always returns the correct value for any $x \in X$.

The Bloomier filter uses a perfect hash function introduced earlier by Czech et al. [5] whose analysis implies that the optimal α is approximately 1.23. A simple calculation shows that in many cases the Bloom map will use less space. In fact it is also straightforward to extend the Bloom map to make use of the same family of perfect hash functions thereby reducing its space requirements to $1.23(\log 1/\epsilon + H(\vec{p}))$.

In the next section we give a complete statement of the problem and prove lower bounds on the space requirements of any data structure supporting approximate queries of a static map with bounded errors (our most general result is Theorem 2). In section 3 we introduce the Simple Bloom map, a data structure supporting approximate queries that has near-optimal space requirements. In section 4 we present more computationally efficient versions of the Bloom map.

2 Problem statement and lower bounds

Consider a map of n key/value pairs $M = \{(x_1, v(x_1)), (x_2, v(x_2)), \dots, (x_n, v(x_n))\}$, where the keys $X = \{x_1, x_2, \dots, x_n\}$ are drawn from a large universe U of size u and each value $v(x)$ is drawn from a fixed set of possible values $V = \{v_1, v_2, \dots, v_b\}$. Suppose further that the distribution of values over keys is given by $\vec{p} = (p_1, p_2, \dots, p_b)$, where $\sum_{i=1}^b p_i = 1$ and $\min_{i \in [b]} p_i > 0$. Thus if $X_i = \{x \in X \mid v(x) = v_i\}$ then $|X_i| = p_i n$. We call such a collection M of key/value pairs a \vec{p} -map.

We consider the problem of constructing a space-efficient data structure supporting queries on a static \vec{p} -map M . For any key $x \in U$ the data structure should return the associated value $v(x)$ if $x \in X$, otherwise it should return $\perp \notin V$. We will be interested in the case when n is large, $u \gg n$ and $b, \vec{p} = (p_1, p_2, \dots, p_b)$ are constant.

Given u, n, b and $\vec{p} = (p_1, p_2, \dots, p_b)$ the total number of distinct \vec{p} -maps is

$$\binom{u}{n} \binom{n}{p_1 n, p_2 n, \dots, p_b n}.$$

By Stirling's formula the multinomial coefficient is $2^{nH(\vec{p}) + O(\log n)}$, where $H(\vec{p}) = -\sum_{i=1}^b p_i \log p_i$ is the entropy of \vec{p} . (Logarithms here and elsewhere are base two.) Hence to distinguish between all \vec{p} -maps without errors we require $m \geq n(\log u - \log n + H(\vec{p}) + o(1))$ bits. For n large and $u \gg n$ this is prohibitive: in particular we require more than a constant number of bits per key. Hence we are obliged to consider lossy data structures.

There are three distinct types of error that we will consider: (*False positives*) $x \in U \setminus X$ is incorrectly assigned a value $v_i \in V$; (*False negatives*) $x \in X_i$ is incorrectly assigned the value \perp ; (*Misassignments*) $x \in X_i$ is incorrectly assigned a value $v \in V \setminus \{v_i\}$.

Let s be a binary string supporting queries by keys $x \in U$, i.e. $s : U \rightarrow V \cup \{\perp\}$. Suppose that we use s to encode a \vec{p} -map M with key set X . We wish to bound the proportion of keys on which

s returns an incorrect value. For $i \in [b]$ we define

$$f^+(s) = \frac{|\{x \in U \setminus X \mid s(x) \neq \perp\}|}{|U \setminus X|},$$

$$f_i^*(s) = \frac{|\{x \in X_i \mid s(x) \in V \setminus \{v_i\}\}|}{|X_i|}, \quad f_i^-(s) = \frac{|\{x \in X_i \mid s(x) = \perp\}|}{|X_i|}.$$

Thus $f^+(s)$ is the proportion of false positives returned on $U \setminus X$, $f_i^*(s)$ is the proportion of misassigned values on X_i and $f_i^-(s)$ is the proportion of false negatives returned on X_i .

Given constants $\epsilon^+ > 0$ and $\epsilon^*, \epsilon^- \geq 0$ we will say that s ($\epsilon^+, \epsilon^*, \epsilon^-$)-encodes M if it satisfies: $f^+(s) \leq \epsilon^+$ and, for all $i \in [b]$, $f_i^*(s) \leq \epsilon^*$ and $f_i^-(s) \leq \epsilon^-$. (We will assume throughout that $\max\{\epsilon^+, \epsilon^*, \epsilon^-\} < 1/8$.)

If the only errors we allow are false positives then we have an $(\epsilon^+, 0, 0)$ -encoding data structure. (An example of such a data structure is the Bloomier filter [3]). Theorem 1 gives lower bounds on the space requirements of such a data structure. (The proof follows a counting argument generalising the argument applied to the approximate set membership problem by Carter et al. [2].)

Theorem 1 *The average number of bits required per key in any data structure that $(\epsilon^+, 0, 0)$ -encodes all \vec{p} -maps is at least*

$$\log 1/\epsilon^+ + H(\vec{p}) + o(1).$$

Proof. Suppose that the m -bit string s $(\epsilon^+, 0, 0)$ -encodes some particular \vec{p} -map M with key set X . For $i \in [b]$ let $A_i^{(s)} = \{x \in U \mid s(x) = v_i\}$, $a_i^{(s)} = |A_i^{(s)}|$ and define $q_i^{(s)}$ by $a_i^{(s)} = p_i n + \epsilon^+(u - n)q_i^{(s)}$. Since $X_i = \{x \in X \mid v(x) = v_i\}$ has size $p_i n$ and s always answers correctly on X_i we have $q_i^{(s)} \geq 0$.

The proportion of $x \in U \setminus X$ for which $s(x) \neq \perp$ is $\sum_{i=1}^b \epsilon^+ q_i^{(s)}$. Since $f^+(s) \leq \epsilon^+$, this implies that $\sum_{i=1}^b q_i^{(s)} \leq 1$.

If N is any \vec{p} -map with key set Y that is also $(\epsilon^+, 0, 0)$ -encoded by s then, since s correctly answers all queries on keys in Y , we have $Y_i = \{y \in Y \mid v(y) = v_i\} \subseteq A_i^{(s)}$, for all $i \in [b]$. Hence, since $|Y_i| = p_i n$, s can $(\epsilon^+, 0, 0)$ -encode at most the following number of distinct \vec{p} -maps

$$\prod_{i=1}^b \binom{a_i^{(s)}}{p_i n} = \prod_{i=1}^b \binom{p_i n + \epsilon^+(u - n)q_i^{(s)}}{p_i n}.$$

Choosing $q_1, q_2, \dots, q_b \geq 0$ to maximise this expression, subject to $\sum_{i=1}^b q_i \leq 1$, we have

$$2^m \prod_{i=1}^b \binom{p_i n + \epsilon^+(u - n)q_i}{p_i n} \geq \binom{u}{n} \binom{n}{p_1 n, p_2 n, \dots, p_b n}.$$

Using the fact that $\frac{(a-b)^b}{b!} \leq \binom{a}{b} \leq \frac{a^b}{b!}$ and taking logarithms we require

$$m + \sum_{i=1}^b p_i n \log(p_i n + \epsilon^+(u - n)q_i) \geq n \log(u - n).$$

Dividing by n , recalling that $\sum_{i=1}^b p_i = 1$ and rearranging we obtain

$$\frac{m}{n} \geq \log 1/\epsilon^+ + \sum_{i=1}^b p_i \log 1/q_i - \sum_{i=1}^b p_i \log \left(1 + \frac{n(p_i - \epsilon^+ q_i)}{\epsilon^+ q_i u} \right) + \log \left(1 - \frac{n}{u} \right).$$

Our assumption that $u \gg n$ (which is equivalent to $n/u = o(1)$) together with the fact that $\log(1 + \alpha) = O(\alpha)$ for α small implies that the last two terms are $o(1)$. Hence the average number of bits required per key satisfies

$$\frac{m}{n} \geq \log 1/\epsilon^+ + \sum_{i=1}^b p_i \log 1/q_i + o(1).$$

Gibbs' inequality implies that the sum is minimised when $q_i = p_i$ for all $i \in [b]$, the result follows. \square

This calculation can be extended to the case when errors are also allowed on keys in the set X .

Theorem 2 *The average number of bits required per key in any data structure that $(\epsilon^+, \epsilon^*, \epsilon^-)$ -encodes all \vec{p} -maps is at least*

$$(1 - \epsilon^-) \log 1/\epsilon^+ + (1 - \epsilon^- - \epsilon^*) H(\vec{p}) - H(\epsilon^-, \epsilon^*, 1 - \epsilon^- - \epsilon^*) + o(1).$$

Proof: The basic idea behind the proof of this result is the same as that of Theorem 1, however the details are somewhat more involved. (See Appendix.) \square

The Bloom map, which we introduce in Section 3, is $(\epsilon, \epsilon, 0)$ -encoding. To enable us to evaluate how far its space requirements are from optimal we give the following simple corollary.

Corollary 3 *The average number of bits required per key in any data structure that $(\epsilon, \epsilon, 0)$ -encodes all \vec{p} -maps is at least*

$$(1 - \epsilon)(\log 1/\epsilon + H(\vec{p}) - (\epsilon + \epsilon^2)) + o(1).$$

Proof: Substitute $\epsilon^+ = \epsilon^* = \epsilon$ and $\epsilon^- = 0$ into Theorem 2 and use $\log(1 - \epsilon) \geq -(\epsilon + \epsilon^2)$. \square

3 The Simple Bloom map

Let M be a \vec{p} -map with key set X . Thus, for $i \in [b]$, $X_i = \{x \in X \mid v(x) = v_i\}$ has size $p_i n$. Our first succinct data structure supporting queries for M is the Simple Bloom map. This is constructed by simply storing the values directly in a Bloom filter.

Let B be an array of size m that is initially empty. For each $i \in [b]$ we choose $k_i \geq 1$ independent random hash functions $h_{i,j} : U \rightarrow [m]$ (we will explain how to set k_1, k_2, \dots, k_b optimally below). To store the key/value pair (x, v_i) we compute $h_{i,j}(x)$ for each $j \in [k_i]$ and set the bits $B[h_{i,j}(x)]$ to one. To query B with a key $x \in U$ we compute $h_{i,j}(x)$ for each $i \in [b], j \in [k_i]$ and set

$$\text{qval}(x) = \left\{ i \in [b] \mid \bigwedge_{j=1}^{k_i} B[h_{i,j}(x)] = 1 \right\}.$$

If $\text{qval}(x) = \emptyset$ we return \perp otherwise we return v_c , where $c = \max \text{qval}(x)$. Note that if $(x, v_i) \in M$ then $i \in \text{qval}(x)$ and so \perp is never returned when querying x , i.e. there are no false negatives. However both false positives and misassignments can occur.

Let $t = n \sum_{i=1}^b p_i k_i$ be the total number of hashes performed during the creation of B . Let ρ be the proportion of bits that remain zero in B . If $f^+(B)$ is the false positive probability of B , i.e. the probability that B returns $v \neq \perp$ for a fixed $x \in U \setminus X$, then

$$f^+(B) = \Pr\{\text{qval}(x) \neq \emptyset\} \leq \sum_{i=1}^b \Pr\{i \in \text{qval}(x)\} = \sum_{i=1}^b (1 - \rho)^{k_i}.$$

If $f_i^*(B)$ is the misassignment probability for B over keys in X_i , i.e. the probability that B returns $v \in V \setminus \{v_i\}$ for a fixed $x \in X_i$, then

$$f_i^*(B) = \Pr\{\max \mathbf{qval}(x) > i\} \leq \sum_{j=i+1}^b \Pr\{j \in \mathbf{qval}(x)\} = \sum_{j=i+1}^b (1-\rho)^{k_j} < \sum_{i=1}^b (1-\rho)^{k_i}.$$

Hence in order to minimise $f^+(B)$ and $f_i^*(B)$ we consider the constrained optimisation problem: minimise $\sum_{i=1}^b (1-\rho)^{k_i}$ subject to $\sum_{i=1}^b p_i k_i = t/n$. A standard application of Lagrange multipliers yields the solution

$$k_i = \frac{t}{n} + \frac{H(\vec{p}) + \log p_i}{\log(1-\rho)}.$$

For this choice of the k_i we have

$$\sum_{i=1}^b (1-\rho)^{k_i} = (1-\rho)^{t/n} 2^{H(\vec{p})} \sum_{i=1}^b p_i = 2^{H(\vec{p})} (1-\rho)^{t/n}.$$

By a simple martingale argument, identical to that given by Mitzenmacher [6] for the Bloom filter, ρ is extremely close to its expected value if $t = O(m)$ (see Appendix). Assuming $\rho \geq \mathbb{E}[\rho]$ we have

$$2^{H(\vec{p})} (1-\rho)^{t/n} \leq 2^{H(\vec{p})} \left(1 - \left(1 - \frac{1}{m}\right)^t\right)^{t/n}.$$

This last expression (without the factor $2^{H(\vec{p})}$) is familiar from the standard Bloom filter error analysis: it is minimised at $t = m \ln 2$, when it equals $2^{H(\vec{p}) - \frac{m}{n} \ln 2}$. (Note that as for the standard Bloom filter the expected proportion of bits set in B is $1/2$.)

Thus to guarantee $f^+(B) \leq \epsilon$ and $f_i^*(B) \leq \epsilon$ for all $i \in [b]$ it is sufficient to take

$$m = n \log e (\log 1/\epsilon + H(\vec{p})), \quad k_i = \log 1/\epsilon + \log 1/p_i \quad \text{for } i \in [b].$$

(As with the standard Bloom filter, the k_i must be integers, for simplicity we will ignore this.)

Since Corollary 3 gives a lower bound for the space required by an $(\epsilon, \epsilon, 0)$ -encoding data structure we would like to claim that B is $(\epsilon, \epsilon, 0)$ -encoding. This is not quite true: the *expected* proportion of false positives and misassignments is at most ϵ but this does not guarantee that B is $(\epsilon, \epsilon, 0)$ -encoding. However B is still essentially $(\epsilon, \epsilon, 0)$ -encoding since, with high probability, the proportion of false positives or misassignments is at most $\epsilon + O(1/\sqrt{n})$. (See Appendix for details.)

Theorem 4 *The Simple Bloom map $(\epsilon, \epsilon, 0)$ -encodes all \vec{p} -maps and uses $\log e (\log 1/\epsilon + H(\vec{p}))$ bits per key.*

Note that by Corollary 3 the space requirements of the Simple Bloom map are essentially a factor $(1-\epsilon)^{-1} \log e$ from optimal, for $\epsilon \leq 0.01$ this is less than 1.46.

We remark that an $(\epsilon, \epsilon, \epsilon)$ -encoding data structure can be created from the Simple Bloom map by simply discarding $\epsilon p_i n$ keys from X_i for each $i \in [b]$. The amount of memory saved is $\epsilon n \log e (\log 1/\epsilon + H(\vec{p}))$ (cf. Theorem 2).

Although the Simple Bloom map is succinct it suffers from two obvious drawbacks if b is not small: the number of hashes/bit probes performed during a query and the number of independent hash functions required is $O(b \log(b/\epsilon))$. In section 4 we explain how to overcome these problems by “reusing” hash functions and using an optimal binary search tree.

Store (x, v_i, T, B)	Query (x, T, B)	Findval (x, v, T, B)
for $d = 0$ to l_{v_i}	$v \leftarrow \perp$	$w \leftarrow r(T)$
$w \leftarrow P_{v_i} \cap T_d$	Findval (x, v, T, B)	for $j = 1$ to k_w
for $j = 1$ to k_w	return v	if $B[h_{w,j}(x)] = 0$ then return false
$B[h_{w,j}(x)] \leftarrow 1$		if w is a leaf then
		$v \leftarrow \text{val}(w)$; return true
		if Findval (x, v, T_R, B) then return true
		return Findval (x, v, T_L, B)

Figure 1: Storing and querying keys in a Bloom map

4 Efficient Bloom maps

Let M be a \vec{p} -map that we wish to store. Sort the list of probabilities of keys so that $p_1 \geq p_2 \geq \dots \geq p_b$. Construct an optimal alphabetic binary tree $T(\vec{p})$ for \vec{p} with leaves labelled v_1, v_2, \dots, v_b (by for example the Garsia–Wachs algorithm, see Knuth [4] page 446). The label of a leaf w is denoted by $\text{val}(w)$. Note that $T(\vec{p})$ is a full binary tree, i.e. every node is either a leaf or has exactly two children.

For any binary tree T let $r(T)$ denote its root and T_L, T_R denote its left and right subtrees respectively. For any node w let P_w denote the set of nodes on the path in T from the root to w and let $l_w = |P_w| - 1$ be the depth of w . For $d \geq 0$ let T_d be the set of nodes in T at depth d .

We number the nodes in $T(\vec{p})$ from left to right at each level, starting at the root and going down. We call these numbers *offsets*. (So the root has offset 0, its left child has offset 1 and its right child has offset 2 etc.) Note that all nodes have distinct offsets. The offset of a node w is denoted $\text{off}(w)$. To each node $w \in T(\vec{p})$ we also associate an integer k_w . We will specify choices for the k_w later, we first impose two simple conditions: $k_w \geq 1$ for all nodes and $k_w \geq \log 1/\epsilon$ for all leaves. Set

$$m = \log e \left(\sum_{i=1}^b p_i n \sum_{w \in P_{v_i}} k_w \right), \quad k = \max_{i \in [b]} \sum_{w \in P_{v_i}} k_w. \quad (1)$$

We now impose a third condition on the k_w : they are chosen so that $m \leq 2n \log e \log(b/\epsilon)$.

Let h_1, h_2, \dots, h_k be independent random hash functions, $h_j : U \rightarrow [m]$. (We will refer to these as the *base* hash functions.) For a node w let $s_w = \sum_{u \in P_w \setminus \{w\}} k_u$. We associate k_w hash functions with w : $h_{w,1}, h_{w,2}, \dots, h_{w,k_w}$, where $h_{w,j} : U \rightarrow [m]$ is defined by $h_{w,j}(x) = h_{s_w+j}(x) + \text{off}(w) \bmod m$.

The Bloom map B is an array of size m that is initially empty (all bits are zero). To store a key/value pair (x, v_i) we use the algorithm **Store**($x, v_i, T(\vec{p}), B$) (see Figure 1). This does the following: for each node w in the path P_{v_i} , starting from the root, it evaluates the associated k_w hashes at x and sets the corresponding bits in B . Note that (ignoring offsets) the hash functions used while storing (x, v_i) are h_1, h_2, \dots, h_{t_i} , where $t_i = \sum_{w \in P_{v_i}} k_w$. Hence the bits which are set in B by **Store**($x, v_i, T(\vec{p}), B$) are chosen independently and uniformly at random. Moreover, since each key is stored with at most one value, the entire process of storing the \vec{p} -map in B is equivalent to setting $t = n \sum_{i=1}^b p_i t_i$ independently chosen random bits in B .

To query B with a key $x \in U$ we use the algorithm **Query**($x, T(\vec{p}), B$). This calls **Findval**($x, v, T(\vec{p}), B$) with v initialised to \perp and returns the value of v when **Findval**($x, v, T(\vec{p}), B$)

terminates (see Figure 1). Starting with $T(\vec{p})$, Findval evaluates the hash functions associated with the root of the current tree, returning false if it finds a zero bit in B , otherwise it continues down the tree, first looking at the right subtree and then, if this fails, looking at the left subtree. If it reaches a leaf at which the corresponding bits in B are all set then v is assigned the value associated with this leaf and it returns true, otherwise the value of v will remain equal to \perp .

By our choice of $m = t \log e$ the expected proportion of bits that remain zero in B (once we have stored the \vec{p} -map M) is $1/2$ and with high probability the actual proportion, which we denote by ρ , is very close to this. For simplicity we will assume that $\rho \geq 1/2$.

We now consider the probability of errors. To simplify our analysis we assume that any leaf v_i is at depth $\log 1/p_i$ (since $T(\vec{p})$ is an optimal alphabetic binary tree this is almost true). For $x \in U$ and $i \in [b]$ define

$$\mathcal{H}_i(x) = \{h_{w,l}(x) \mid w \in P_{v_i}, l \in [k_w]\}, \quad \text{qval}(x) = \{i \in [b] \mid \bigwedge_{h \in \mathcal{H}_i(x)} B[h] = 1\}.$$

Thus $i \in \text{qval}(x)$ iff all of the bits in B indexed by the hash functions on the path P_{v_i} evaluated at x are set. If $\text{qval}(x) = \emptyset$ then Query returns \perp , otherwise, since Findval always explores right subtrees first, it returns v_c , where $c = \max \text{qval}(x)$. If $x \in X_i$ then $i \in \text{qval}(x)$ and so no false negatives can occur. False positives and misassignments are possible, we consider the case of false positives first.

If $x \in U \setminus X$ then for fixed $i \in [b]$ the bits in $\mathcal{H}_i(x)$ are simply independent random choices from $[m]$. This is because if $t_i = \sum_{w \in P_{v_i}} k_w$ then the hash functions we evaluate are simply offsets, modulo m , of the first t_i of our base hash functions. By our assumptions that: $k_w \geq 1$ for all nodes; $k_{v_i} \geq \log 1/\epsilon$ and v_i is at depth $\log 1/p_i$, we have $t_i \geq -\log \epsilon p_i$. Since $\rho \geq 1/2$ the false positive probability satisfies

$$f^+(B) = \Pr\{\text{qval}(x) \neq \emptyset\} \leq \sum_{i=1}^b \Pr\{i \in \text{qval}(x)\} \leq \sum_{i=1}^b (1 - \rho)^{t_i} \leq \sum_{i=1}^b \frac{1}{2^{t_i}} \leq \epsilon.$$

Calculating the probability of a misassignment when B is queried with $x \in X_i$ is more involved. Note that if an incorrect value $v_j \neq v_i$ is returned for $x \in X_i$ then $j > i$. For $i < j$ and $x \in X_i$ let $P_{i,j} = P_{v_j} \setminus P_{v_i}$ be the part of the path P_{v_j} that is disjoint from the path P_{v_i} and let $\mathcal{H}_{i,j}(x) = \{h_{w,l}(x) \mid w \in P_{i,j}, l \in [k_w]\}$. The misassignment probability satisfies

$$f_i^*(B) = \Pr\{\max \text{qval}(x) > i\} \leq \sum_{j=i+1}^b \Pr\{j \in \text{qval}(x)\} = \sum_{j=i+1}^b \Pr\left\{\bigwedge_{h \in \mathcal{H}_{i,j}(x)} B[h] = 1\right\}. \quad (2)$$

To bound this probability we consider the following: suppose that rather than storing all of the key/value pairs from M in B we had instead stored all of them *except* (x, v_i) . Let B' denote the resulting m -bit array. Let $t_{i,j} = |\mathcal{H}_{i,j}(x)|$. Since (x, v_i) has not been stored in B' we have (by the same argument as used for $f^+(B)$) that

$$\Pr\left\{\bigwedge_{h \in \mathcal{H}_{i,j}(x)} B'[h] = 1\right\} \leq \frac{1}{2^{t_{i,j}}}. \quad (3)$$

If all of the bits in B indexed by elements in $\mathcal{H}_{i,j}$ are set then either they are all set in B' or there must be at least one bit in $\mathcal{H}_{i,j}$ that is only set once (x, v_i) is stored. The later case can only occur if $\mathcal{H}_i \cap \mathcal{H}_{i,j} \neq \emptyset$. Hence

$$\Pr\left\{\bigwedge_{h \in \mathcal{H}_{i,j}(x)} B[h] = 1\right\} \leq \Pr\left\{\bigwedge_{h \in \mathcal{H}_{i,j}(x)} B'[h] = 1\right\} + \Pr\{\mathcal{H}_i \cap \mathcal{H}_{i,j} \neq \emptyset\}. \quad (4)$$

If $\hat{h}_1 \in \mathcal{H}_i(x)$ and $\hat{h}_2 \in \mathcal{H}_{i,j}(x)$ then $\Pr\{\hat{h}_1 = \hat{h}_2\}$ is either $1/m$ or 0, since \hat{h}_1 and \hat{h}_2 either use different base hash functions (and so are independent and random in $[m]$) or they use the same base hash function with different offsets and hence are distinct.

Recall that $k = \max_{i \in [b]} \sum_{w \in P_{v_i}} k_w$. If $c \in [b]$ satisfies $k = \sum_{w \in P_{v_c}} k_w$ then $m \geq np_c k \log e$. Moreover the k_w were chosen so that $n \leq m \leq 2n \log e \log(b/\epsilon)$. Hence

$$\Pr\{\mathcal{H}_i \cap \mathcal{H}_{i,j} \neq \emptyset\} \leq \frac{|\mathcal{H}_i| \cdot |\mathcal{H}_{i,j}|}{m} \leq \frac{k^2}{m} \leq \left(\frac{m}{np_c \log e}\right)^2 \frac{1}{m} \leq \left(\frac{2 \log b/\epsilon}{p_c}\right)^2 \frac{1}{n} = O\left(\frac{1}{n}\right), \quad (5)$$

where the final equality uses our assumption that \vec{p} , b and ϵ are constant.

Combining (2), (3), (4) and (5) we obtain

$$f_i^*(B) \leq \sum_{j=i+1}^b \frac{1}{2^{t_{i,j}}} + O\left(\frac{1}{n}\right) \approx \sum_{j=i+1}^b \frac{1}{2^{t_{i,j}}}, \quad (6)$$

where $t_{i,j} = \sum_{w \in P_{i,j}} k_w$. Thus to ensure $f_i^*(B) \leq \epsilon$ we choose the k_w so that $\sum_{j=i+1}^b 2^{-t_{i,j}} \leq \epsilon$. There are various ways in which this can be done and exactly how we choose the k_w will effect not only $f_i^*(B)$ but also the memory required to store the Bloom map and the amount of work we expect to do when querying it. Since different space/time trade-offs may be of interest in different applications we define two special types of Bloom map: Standard and Fast.

- (*Standard*) $k_w = 1$ for all internal nodes (i.e. all non-leaf nodes), $k_{v_i} = \log 1/\epsilon + \log(H_b - 1) + 1$ for all leaves (where $H_b = \sum_{l=1}^b 1/l$ is the b th Harmonic number).
- (*Fast*) $k_w = 2$ for all internal nodes, $k_{v_i} = \log 1/\epsilon + 2$ for all leaves.

Theorem 5 *The Standard and Fast Bloom maps are both $(\epsilon, \epsilon, 0)$ -encoding for \vec{p} -maps. The average number of bits required per key is:*

- (*Standard*): $\log e(\log 1/\epsilon + H(\vec{p}) + \log(H_b - 1) + 1)$.
- (*Fast*): $\log e(\log 1/\epsilon + 2H(\vec{p}) + 2)$.

If $x \in U \setminus X$ then the expected number of bit probes performed during $\text{Query}(x, T(\vec{p}), B)$ is at most: (*Standard*) $H(\vec{p}) + 2$; (*Fast*) 3.

If $x \in X_i$ then the expected number of bit probes performed during $\text{Query}(x, T(\vec{p}), B)$ is at most: (*Standard*) $O((\log b)^2) + \log 1/p_i + \log 1/\epsilon$; (*Fast*) $3 \log(b - i + 1) + 2 \log 1/p_i + \log 1/\epsilon + 2$.

The Standard Bloom map uses little more than a factor $(1 - \epsilon)^{-1} \log e$ extra bits per key than the lower bound of Corollary 3. (In addition to the factor of $(1 - \epsilon)^{-1} \log e$ it uses at most an extra $1 + \log \log b$ bits per key, since $H_b < \log b$.) The Fast Bloom map uses slightly more space but has the advantage of using significantly fewer bit probes when querying keys: in particular we expect to perform at most 3 bit probes on $x \in U \setminus X$. In any case the Fast Bloom map uses less than 2.9 times as much memory per key as the lower bound and if $H(\vec{p})$ is small compared to $\log 1/\epsilon$ this factor will be much closer to 1.46.

We note that other choices for the k_w are possible and depending on the application may be desirable. For example, altering the Fast Bloom map by adding $s \geq 1$ to k_r , where r is the root of $T(\vec{p})$, yields a Bloom map that will perform $2 + 1/2^s$ bit probes on average, for $x \in U \setminus X$. Another

possibility is to alter the Standard Bloom map by adding $\log(H(\vec{p}) + 2)$ to the value of k_r giving a Bloom map which performs the same expected number of bit probes as the Fast Bloom map on $x \in U \setminus X$ and the same expected number of bit probes as the Standard Bloom map on $x \in X$.

Proof of Theorem 5: We first show that both Bloom maps are $(\epsilon, \epsilon, 0)$ -encoding. We know already that $f^+(B) \leq \epsilon$ so we consider $f_i^*(B)$. We require the following simple lemma.

Lemma 6 *Let T be a full binary tree with leaves v_1, v_2, \dots, v_b at depths $l_1 \leq l_2 \leq \dots \leq l_b$.*

(a) *If $1 \leq i < j \leq b$ then the number of nodes in $P_{v_j} \setminus P_{v_i}$ is at least $\log \left(\sum_{k=i}^j 2^{l_j - l_k} \right)$.*

(b) *If T_d is the set of nodes in T at depth d then*

$$\sum_{d=0}^{l_b} \frac{|T_d|}{2^d} \leq 1 + \sum_{i=1}^b \frac{l_i}{2^{l_i}}.$$

(c) *The number of left branches on the path P_{v_i} is at most $\log(b - i + 1)$.*

Proof: These are all straightforward, see Appendix for details. \square

Lemma 6 (a), together with our assumption that v_k is at depth $\log 1/p_k$ in $T(\vec{p})$ and the fact that $p_1 \geq p_2 \geq \dots \geq p_b$ implies that the number of internal nodes on $P_{i,j}$ is at least $\log \left(\sum_{j=i}^k p_k/p_j \right) - 1$. Let a be the common value of k_w for all internal nodes. By (6) we have

$$f_i^*(B) \leq \sum_{j=i+1}^b \frac{1}{2^{t_{i,j}}} \leq \sum_{j=i+1}^b \left(\frac{p_j}{\sum_{k=i}^j p_k} \right)^a \frac{1}{2^{k_{v_j} - a}} \leq \sum_{j=i+1}^b \frac{1}{(j - i + 1)^a 2^{k_{v_j} - a}},$$

where the last inequality follows from the fact that $p_j \leq p_k$ for all $i \leq k \leq j$. In the case of the Standard Bloom map we have $a = 1$ and $k_{v_j} = \log 1/\epsilon + \log(H_b - 1) + 1$, hence $f_i^*(B) \leq \epsilon$. For the Fast Bloom map $a = 2$, $k_{v_j} = \log 1/\epsilon + 2$ and $\sum_{k=1}^{\infty} 1/l^2 = \pi^2/6$ imply that

$$f_i^*(B) \leq \sum_{l=2}^{b-i+1} \frac{\epsilon}{l^2} \leq \epsilon \left(\frac{\pi^2}{6} - 1 \right) < \epsilon.$$

Hence both Bloom maps are $(\epsilon, \epsilon, 0)$ -encoding.

Now consider how much work we expect to do when querying B . We measure this in terms of the expected number of bit probes performed. (Note that as described each bit probe performed by **Findval** involves the evaluation of a hash function, this need not be the case. The use of offsets ensures that we never need to evaluate more than $k = \max_{i \in [b]} \sum_{i \in P_{v_i}} k_w$ base hash functions, different offsets can then be added as required.) We consider the cases $x \in X$, $x \in U \setminus X$ separately.

Let negbp denote the expected number of bit probes performed by **Query**($x, T(\vec{p}), B$) for $x \in U \setminus X$. The easiest case is the Fast Bloom map, in which every internal node w has $k_w = 2$. Let $\text{negbp}(T)$ be the expected number of bit probes performed by **Findval** in a tree T . We wish to find $\text{negbp} = \text{negbp}(T(\vec{p}))$. Starting from the root of $T(\vec{p})$ we have

$$\text{negbp}(T(\vec{p})) \leq 1 + \frac{1}{2} + \frac{1}{4}(\text{negbp}(T_L(\vec{p})) + \text{negbp}(T_R(\vec{p}))),$$

since if b_1, b_2 are the first two bit probes then $\Pr\{b_1 = 0\} = \rho \geq 1/2$ and $\Pr\{b_1 = b_2 = 1\} = (1 - \rho)^2 \leq 1/4$. Iterating and using the fact that all nodes in $T(\vec{p})$ have at least two associated bit probes we find

$$\text{negbp} \leq \frac{3}{2} \sum_{j=0}^{\infty} \frac{1}{2^j} = 3.$$

In the Standard Bloom map $k_w = 1$ for every internal node, hence if w is at depth l_w then the probability that the bit probe associated with w is evaluated during $\text{Query}(x, T(\vec{p}), B)$, is at most 2^{-l_w} . Moreover for a leaf v_i at depth $\log 1/p_i$ the probability that Findval performs more than one bit probe at v_i is at most $p_i/2$ and in this case we expect to perform at most two extra bit probes at the leaf. Hence if $T_d(\vec{p})$ is the set of nodes in $T(\vec{p})$ at depth d then the expected number of bit probes performed during $\text{Query}(x, T(\vec{p}), B)$ is at most

$$\text{negbp} \leq 2 \sum_{i=1}^b \frac{p_i}{2} + \sum_{d=0}^{\infty} \frac{|T_d(\vec{p})|}{2^d} = 1 + \sum_{d=0}^{\infty} \frac{|T_d(\vec{p})|}{2^d}.$$

By Lemma 6 (b) this is at most $H(\vec{p}) + 2$.

Finally we calculate the expected number of bit probes performed by $\text{Query}(x, T(\vec{p}), B)$, for $x \in X_i$, which we denote by $\text{posbp}(i)$. This will be the number of bits set during $\text{Store}(x, v_i, T(\vec{p}), B)$, plus the expected number of bit probes performed by Findval in the “false subtrees” it explores, where a false subtree is any maximal subtree disjoint from the path P_{v_i} . The number of false subtrees is simply the number of left branches in the path P_{v_i} , since at each such branch Findval first explores the right (false) subtree. By Lemma 6 (c) the number of false subtrees is at most $\log(b - i + 1)$. To simplify our analysis we will assume that the bit probes in false subtrees are independent and random. By a similar argument to that used during the calculation of the misassignment probability above this is essentially true.

For the Fast Bloom map we expect to perform at most three bit probes in each false subtree. Since the number of false subtrees in $T(\vec{p})$ is at most $\log(b - i + 1)$ the expected number of bit probes performed in false subtrees is at most $3 \log(b - i + 1)$. Since the number of bits set by $\text{Store}(x, v_i, T(\vec{p}), B)$ is $2 \log 1/p_i + \log 1/\epsilon + 2$ we have

$$\text{posbp}(i) \leq 3 \log(b - i + 1) + 2 \log 1/p_i + \log 1/\epsilon + 2.$$

Now consider the Standard Bloom map. Any false subtree is a full binary tree with $z \leq b - i$ leaves and hence corresponds to an optimal binary search tree for some probability distribution $q = (q_1, q_2, \dots, q_z)$. Since $H(q) \leq \log z \leq \log(b - i)$ the expected number of bit probes performed in any false subtree is at most $\log(b - i) + 2$. The number of bits set by $\text{Store}(x, v_i, T(\vec{p}), B)$ is $\log 1/p_i + \log 1/\epsilon + \log(H_b - 1) + 1$. Hence

$$\text{posbp}(i) = O((\log b)^2) + \log 1/p_i + \log 1/\epsilon.$$

This completes the proof of Theorem 5. □

References

- [1] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

- [2] Larry Carter, Robert Floyd, John Gill, George Markowsky, and Mark Wegman. Exact and approximate membership testers. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 59–65, New York, NY, USA, 1978. ACM Press.
- [3] Bernard Chazelle, Joe Kilian, Ronitt Rubinfeld, and Ayellet Tal. The Bloomier filter: an efficient data structure for static support lookup tables. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 30–39, 2004.
- [4] Donald E. Knuth. *Volume 3 The Art of Computer Programming: Sorting and Searching*. Addison-Wesley, 1998.
- [5] B. S. Majewski¹, N. C. Wormald, G. Havas, and Z. J. Czech. A family of perfect hashing methods. *British Computer Journal*, 39(6):547–554, 1996.
- [6] Michael Mitzenmacher. Compressed Bloom filters. *IEEE/ACM Trans. Netw.*, 10(5):604–612, 2002.

Appendix

Proof of Theorem 2. Let M be a fixed \vec{p} -map with key set X . Suppose that M is $(\epsilon^+, \epsilon^*, \epsilon^-)$ -encoded by the m -bit string s . For $i \in [b]$ let $a_i^{(s)} = |\{x \in U \mid s(x) = v_i\}|$ and let $w_i^{(s)} = |\{x \in U \setminus X \mid s(x) = v_i\}|$. Define $q_i^{(s)} \geq 0$ by $w_i^{(s)} = \epsilon^+(u - n)q_i^{(s)}$. So $a_i^{(s)} \leq n + \epsilon^+(u - n)q_i^{(s)}$. Since $f^+(s) \leq \epsilon^+$ we have

$$\sum_{i=1}^b q_i^{(s)} \leq 1. \quad (7)$$

We now need to consider how many distinct \vec{p} -maps $N = \{(y_1, v(y_1)), (y_2, v(y_2)), \dots, (y_n, v(y_n))\}$ can be $(\epsilon^+, \epsilon^*, \epsilon^-)$ -encoded by the string s . Let Y be the key set of N and for $i \in [b]$ let $Y_i = \{y \in Y \mid v(y) = v_i\}$, so $|Y_i| = p_i n$. For $0 \leq j \leq b$ let $y_{i,j} = |\{y \in Y_i \mid s(y) = v_j\}|$.

Since $f_i^-(s) \leq \epsilon^-$, $f_i^*(s) \leq \epsilon^*$ and s returns a value from $V \cup \{\perp\}$ for each element in Y_i we have the following three constraints on the $y_{i,j}$

$$y_{i,0} \leq \epsilon^- p_i n, \quad \sum_{j \in [b] \setminus \{i\}} y_{i,j} \leq \epsilon^* p_i n, \quad \sum_{j=0}^b y_{i,j} = p_i n. \quad (8)$$

We can now bound the number of choices for the $y_{i,j}$. Since $\sum_{j=0}^b y_{i,j} = p_i n$, $y_{i,i}$ is determined by fixing the values of $y_{i,j}$ for $j \neq i$. Hence the number of choices for the $y_{i,j}$ is at most

$$\epsilon^- p_i n \sum_{l=0}^{\lfloor \epsilon^* p_i n \rfloor} \binom{l + b - 2}{b - 2} \leq \epsilon^- \epsilon^* (p_i n)^2 \binom{\epsilon^* p_i n + b - 2}{b - 2}.$$

(This is because (8) implies that there are at most $\epsilon^- p_i n$ choices for $y_{i,0}$ while $\sum_{j \in [b] \setminus \{i\}} y_{i,j} = l$ for some integer $0 \leq l \leq \epsilon^* p_i n$. The number of ways of choosing $b - 1$ non-negative integers whose sum is l is $\binom{l + b - 2}{b - 2}$.) For a particular choice of the $y_{i,j}$ the number of choices for the keys in Y_i is at most

$$\binom{u}{y_{i,0}} \prod_{j=1}^b \binom{n + \epsilon^+(u - n)q_j^{(s)}}{y_{i,j}}. \quad (9)$$

(This is because any particular choice for the keys in Y_i is given by choosing $y_{i,0}$ keys on which s returns \perp and then choosing $y_{i,j}$ keys on which s returns v_j , for each $j \in [b]$.)

Let $y'_{i,0}, y'_{i,1}, \dots, y'_{i,b}$ be chosen to maximise (9) subject to (8). The number of choices for the keys in Y_i is at most

$$\epsilon^- \epsilon^* (p_i n)^2 \binom{\epsilon^* p_i n + b - 2}{b - 2} \binom{u}{y'_{i,0}} \prod_{j=1}^b \binom{n + \epsilon^+(u - n) q_j^{(s)}}{y'_{i,j}}.$$

Hence the total number of \vec{p} -maps which can be $(\epsilon^+, \epsilon^*, \epsilon^-)$ -encoded by the string s is at most

$$\prod_{i=1}^b \left(\epsilon^- \epsilon^* (p_i n)^2 \binom{\epsilon^* p_i n + b - 2}{b - 2} \binom{u}{y'_{i,0}} \prod_{j=1}^b \binom{n + \epsilon^+(u - n) q_j^{(s)}}{y'_{i,j}} \right).$$

Letting $q_1, q_2, \dots, q_b \geq 0$ be chosen to maximise this expression subject to $\sum_{j=1}^b q_j \leq 1$ we obtain

$$2^m \prod_{i=1}^b \left(\epsilon^- \epsilon^* (p_i n)^2 \binom{\epsilon^* p_i n + b - 2}{b - 2} \binom{u}{y'_{i,0}} \prod_{j=1}^b \binom{n + \epsilon^+(u - n) q_j}{y'_{i,j}} \right) \geq \binom{u}{n} \binom{n}{p_1 n, \dots, p_b n}.$$

Using $\frac{(a-b)^b}{b!} \leq \binom{a}{b} \leq \frac{a^b}{b!} \leq a^b$ we require

$$2^m \prod_{i=1}^b \left(\epsilon^- \epsilon^* (p_i n)^b \left(1 + \frac{b-2}{\epsilon^* p_i n} \right)^b \binom{p_i n}{y'_{i,0}, y'_{i,1}, \dots, y'_{i,b}} u^{y'_{i,0}} \prod_{j=1}^b (\epsilon^+ q_j u)^{y'_{i,j}} \left(1 + \frac{n(1 - \epsilon^+ q_j)}{\epsilon^+ q_j u} \right)^{y'_{i,j}} \right) \geq u^n \left(1 - \frac{n}{u} \right)^n.$$

Taking logarithms and using

$$\sum_{i=1}^b \sum_{j=0}^b y'_{i,j} = \sum_{i=1}^b p_i n = n$$

we obtain

$$\begin{aligned} m \geq & -b \log(\epsilon^- \epsilon^*) - \sum_{i=1}^b b \log(p_i n) - \sum_{i=1}^b b \log \left(1 + \frac{b-2}{\epsilon^* p_i n} \right) - \sum_{i=1}^b p_i n H \left(\frac{y'_{i,0}}{p_i n}, \frac{y'_{i,1}}{p_i n}, \dots, \frac{y'_{i,b}}{p_i n} \right) \\ & + \sum_{i=1}^b \sum_{j=1}^b y'_{i,j} \log(1/\epsilon^+ q_j) - \sum_{i=1}^b \sum_{j=1}^b y'_{i,j} \log \left(1 + \frac{n(1 - \epsilon^+ q_j)}{\epsilon^+ q_j u} \right) + n \log \left(1 - \frac{n}{u} \right). \end{aligned}$$

Defining $r_{i,j} = y'_{i,j}/p_i n$; noting that the first three terms in the previous inequality are all $O(\log n)$ and using $\log(1 + \alpha) = O(\alpha)$ for α small we obtain

$$m \geq n \sum_{i=1}^b p_i \left(\sum_{j=1}^b r_{i,j} \log(r_{i,j}/\epsilon^+ q_j) + r_{i,0} \log r_{i,0} \right) + O(\log n) + O\left(\frac{n^2}{u}\right).$$

Dividing by n and using $u \gg n$ we find that the average number of bits required per key is at least

$$\begin{aligned} \frac{m}{n} &\geq \sum_{i=1}^b p_i \left((1 - r_{i,0}) \log 1/\epsilon^+ + r_{i,0} \log r_{i,0} + r_{i,i} \log r_{i,i} + r_{i,i} \log 1/q_i \right) \\ &\quad + \sum_{i=1}^b \sum_{j \in [b] \setminus \{i\}} p_i r_{i,j} \log r_{i,j} + \sum_{i=1}^b \sum_{j \in [b] \setminus \{i\}} p_i r_{i,j} \log 1/q_j + o(1). \end{aligned} \quad (10)$$

Defining $t_{i,j} = r_{i,j}/(1 - r_{i,0} - r_{i,i})$ we have

$$\sum_{i=1}^b \sum_{j \in [b] \setminus \{i\}} p_i r_{i,j} \log r_{i,j} = \sum_{i=1}^b p_i (1 - r_{i,0} - r_{i,i}) (\log(1 - r_{i,0} - r_{i,i}) - H(t_{i,1}, \dots, t_{i,i-1}, t_{i,i+1}, \dots, t_{i,b})). \quad (11)$$

Defining $u_{i,j} = q_j/(1 - q_i)$ and applying Gibbs' inequality we obtain

$$\begin{aligned} \sum_{i=1}^b \sum_{j \in [b] \setminus \{i\}} p_i r_{i,j} \log 1/q_j &= \sum_{i=1}^b p_i (1 - r_{i,0} - r_{i,i}) \left(\log 1/(1 - q_i) + \sum_{j \in [b] \setminus \{i\}} t_{i,j} \log 1/u_{i,j} \right) \\ &\geq \sum_{i=1}^b p_i (1 - r_{i,0} - r_{i,i}) (\log 1/(1 - q_i) + H(t_{i,1}, \dots, t_{i,i-1}, t_{i,i+1}, \dots, t_{i,b})) \end{aligned} \quad (12)$$

Substituting (11) and (12) into (10) yields

$$\begin{aligned} \frac{m}{n} &\geq \sum_{i=1}^b p_i \left((1 - r_{i,0}) \log 1/\epsilon^+ + r_{i,0} \log r_{i,0} + r_{i,i} \log r_{i,i} + r_{i,i} \log 1/q_i \right) \\ &\quad + \sum_{i=1}^b p_i (1 - r_{i,0} - r_{i,i}) (\log 1/(1 - q_i) + \log(1 - r_{i,0} - r_{i,i})) + o(1). \end{aligned}$$

Defining $r_i^* = \sum_{j \in [b] \setminus \{i\}} r_{i,j}$ we have (by (8)) that $r_i^* \leq \epsilon^*$. We also have $r_{i,0} \leq \epsilon^-$ and so $r_{i,i} = 1 - r_{i,0} - r_i^* \geq 1 - \epsilon^- - \epsilon^*$. Hence

$$\begin{aligned} \frac{m}{n} &\geq (1 - \epsilon^-) \log 1/\epsilon^+ - H(\epsilon^-, \epsilon^*, 1 - \epsilon^- - \epsilon^*) + \sum_{i=1}^b p_i (r_{i,i} \log 1/q_i + r_i^* \log 1/(1 - q_i)) + o(1) \\ &\geq (1 - \epsilon^-) \log 1/\epsilon^+ - H(\epsilon^-, \epsilon^*, 1 - \epsilon^- - \epsilon^*) + (1 - \epsilon^- - \epsilon^*) \sum_{i=1}^b p_i \log 1/q_i \\ &\quad + \sum_{i=1}^b p_i r_i^* \log 1/(1 - q_i) + o(1). \end{aligned} \quad (13)$$

Finally applying Gibbs' inequality and noting that the last summation in (13) is non-negative yields our desired lower bound on the average number of bits required per key

$$\frac{m}{n} \geq (1 - \epsilon^-) \log 1/\epsilon^+ + (1 - \epsilon^- - \epsilon^*) H(\vec{p}) - H(\epsilon^-, \epsilon^*, 1 - \epsilon^- - \epsilon^*) + o(1).$$

□

Justification that ρ , the proportion of zeros in a Simple Bloom map, is sharply concentrated.

If Y_j is the expected number of bits that remain zero in the Simple Bloom map B , conditioned on the first j hashes then $Y_0 = E[\rho m]$ while $Y_t = \rho m$. The Y_j form a martingale with $|Y_{j+1} - Y_j| \leq 1$. Azuma's inequality now implies that for any $\lambda > 0$ we have

$$\Pr \left\{ \rho < E[\rho] - \frac{\lambda \sqrt{t}}{m} \right\} < e^{-\lambda^2/2}.$$

Hence if $t = O(m)$ then ρ is extremely unlikely to be much smaller than its expected value. (Note that this argument also implies that the same is true for the more efficient Bloom maps described in Section 4.)

Remark on $(\epsilon, \epsilon, 0)$ -encoding. Having given lower bounds on the space required by $(\epsilon, \epsilon, 0)$ -encoding data structures in Corollary 3 we would like to claim that the Simple Bloom map B is $(\epsilon, \epsilon, 0)$ -encoding. This is not quite true: the *expected* proportion of false positives and misassignments is at most ϵ but this does not guarantee that B is $(\epsilon, \epsilon, 0)$ -encoding. (This is no different from the often overlooked fact that for an ordinary Bloom filter with false positive probability ϵ the *proportion* of keys in $U \setminus X$ for which the filter returns a false positive may be larger than ϵ .) However the events “ B returns a false positive on query x ”, $x \in U \setminus X$, are independent and have probability at most $f^+(B)$. Hence if Z is the number of false positives in $U \setminus X$ then Z is stochastically dominated by the binomially distributed variable $\text{Bin}(u - n, f^+(B))$. Using Hoeffding's bound for the tail of the binomial distribution we have

$$\Pr\{Z > (u - n)f^+(B) + \lambda\sqrt{u - n}\} \leq e^{-\lambda^2/2}.$$

Hence with high probability the proportion of false positives is at most $f^+(B) + O(1/\sqrt{u - n})$. Similarly the proportion of misassignments is (with high probability) at most $f_i^*(B) + O(1/\sqrt{n})$. Thus B is essentially $(\epsilon, \epsilon, 0)$ -encoding. (Note that a similar argument implies that this also holds for the more efficient Bloom maps of Section 4.)

Proof of Lemma 6. First note that if T is a perfect binary tree (i.e. a full binary tree with all leaves at the same depth) then the number of nodes on $P_{i,j}$, (where $P_{i,j}$ is the part of the path from the root to v_j that is disjoint from the path to v_i), is at least $\log(j - i + 1)$.

Now extend the tree T to a tree T' by replacing each leaf $v_k \in \{v_i, v_{i+1}, \dots, v_{j-1}\}$ by a perfect binary tree of depth $l_j - l_k$. By our previous remark the number of nodes on $P_{i,j}$ is at least $\log s$, where s is the number of leaves lying strictly between v_{i-1} and v_{j+1} in T' . Since $s = \sum_{k=i}^j 2^{l_j - l_k}$ part (a) now follows.

For (b) note that if we define $l_0 = 0$ then

$$\begin{aligned} \sum_{d=0}^{l_b} \frac{|T_d|}{2^d} &\leq 1 + \sum_{i=1}^b (l_i - l_{i-1}) \left(1 - \sum_{j=1}^i \frac{1}{2^{l_j}} \right) \\ &= 1 + \sum_{i=1}^b \frac{l_i}{2^{l_i}}. \end{aligned}$$

For (c) note that if the path from the root to v_i has left branches at depths d_1, d_2, \dots, d_t then the number of leaves to the right of v_i is at least $\sum_{j=1}^t 2^{l_i - (d_j + 1)}$ (this is because T is full). Since

all of the depths of the left branches are distinct and at most $l_i - 1$, the number of leaves to the right of v_i is at least $\sum_{j=0}^{t-1} 2^j = 2^t - 1$. However the number of leaves to the right of v_i is $b - i$ and so $t \leq \log(b - i + 1)$. \square